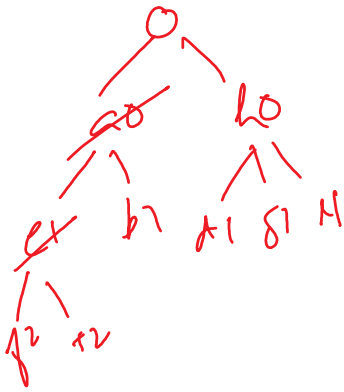
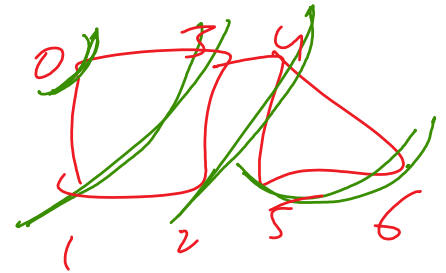
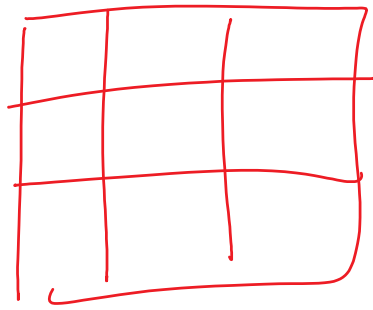
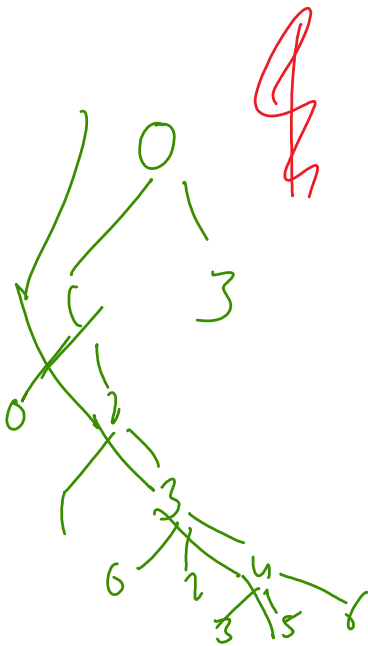
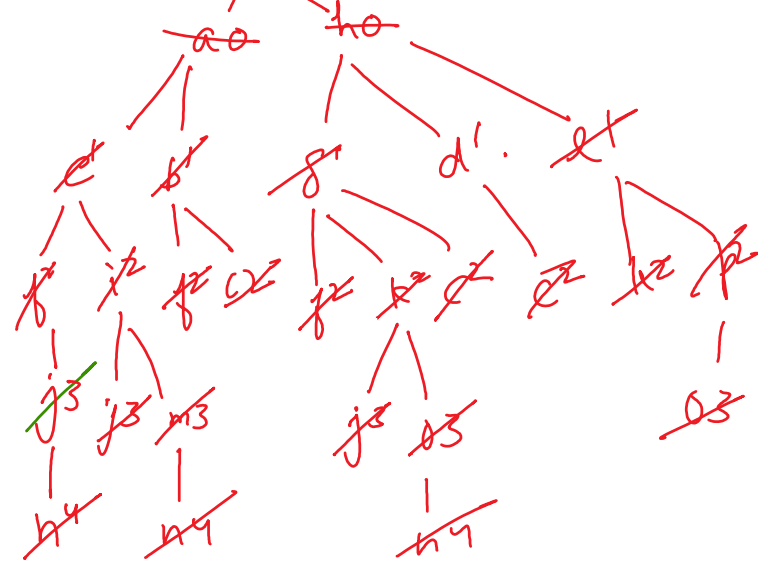


DP → smallest → largest



a	b	c	d
0	1	2	1
e 1	f 2	g 1	h 0
i 2	j 3	k 2	l 1
m 3	n 4	o 3	p 2



DP

0	.1	1	1
.1	1	1	1
1	1	1	1
1	1	1	1

0	1	2	.
8		.	0

DP \rightarrow Topological

$$V \propto \Sigma$$
$$\sqrt{75}$$
$$\underline{n \times m} + 4(n \times m)$$

ham

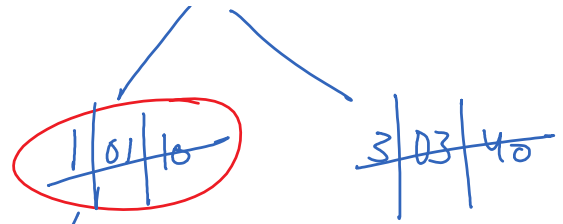
A 5x5 grid of numbers from 0 to 24, arranged in a snake-like pattern. Blue arrows indicate a path starting at 0, moving right to 1, then up to 2, then right to 3, then down to 4, and finally right to 5.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

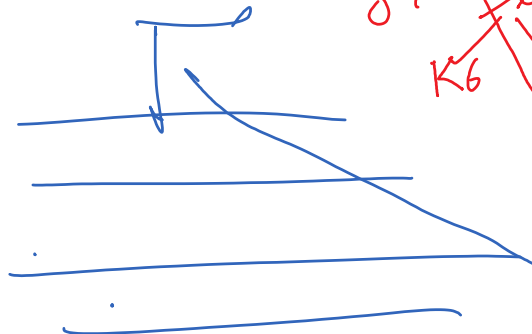
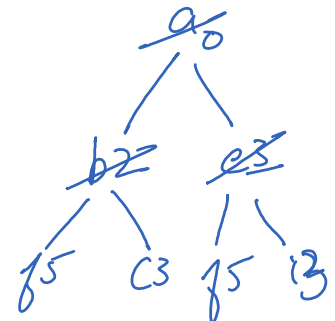
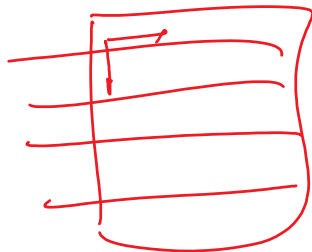
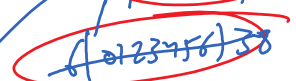
$$1 - 1$$

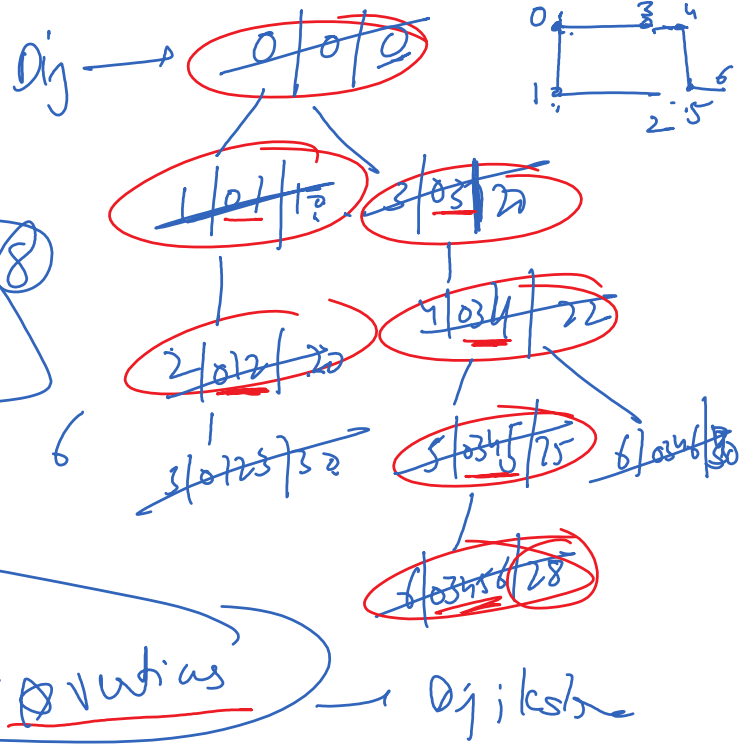
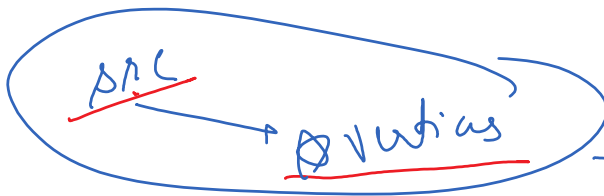
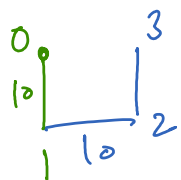
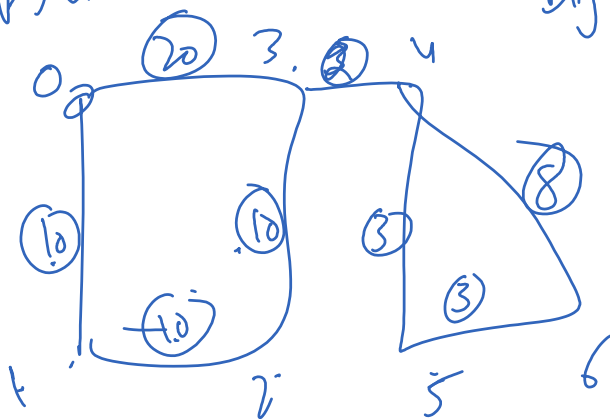
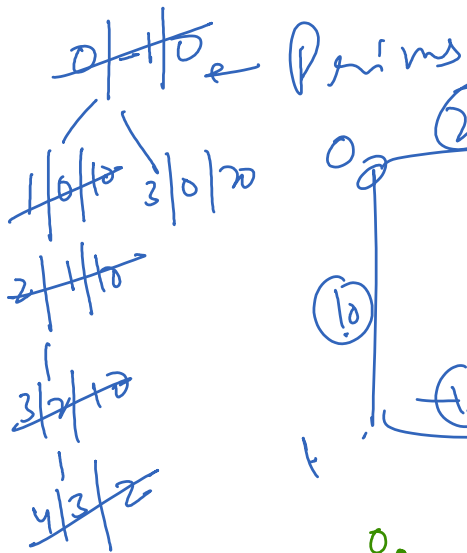
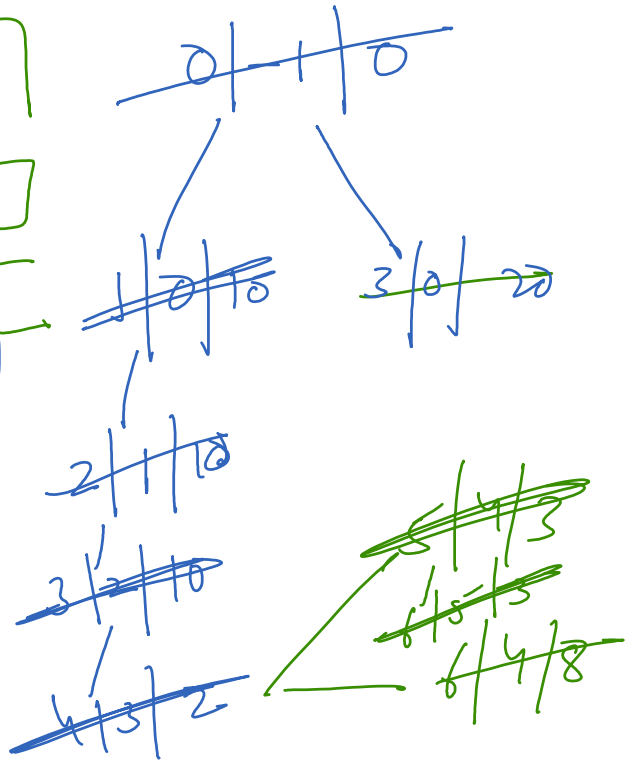
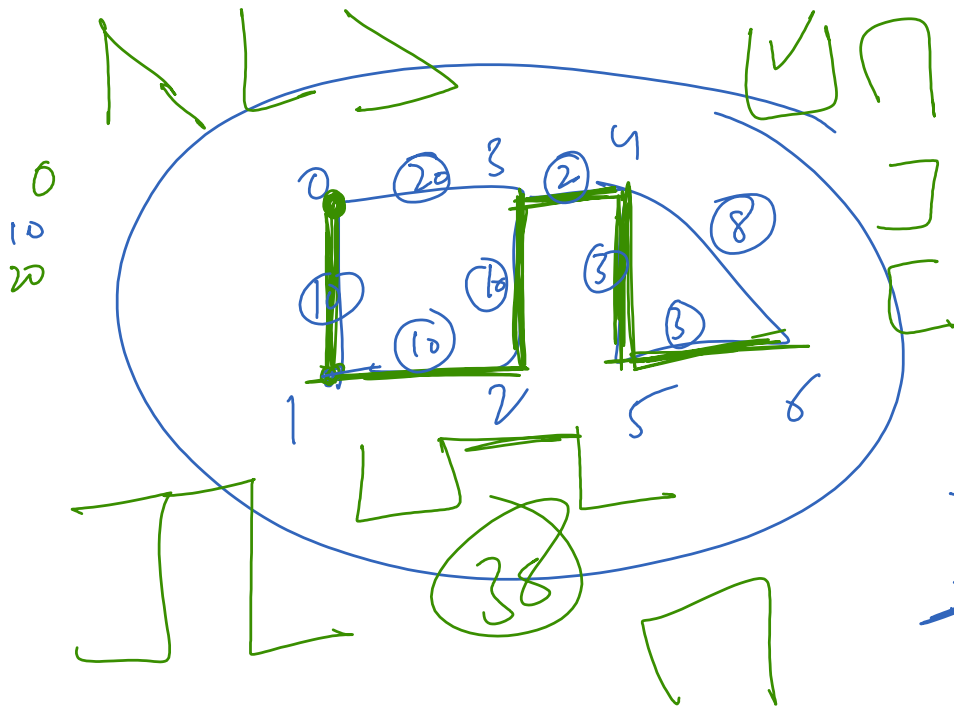
$x: 16 - 10; 30$

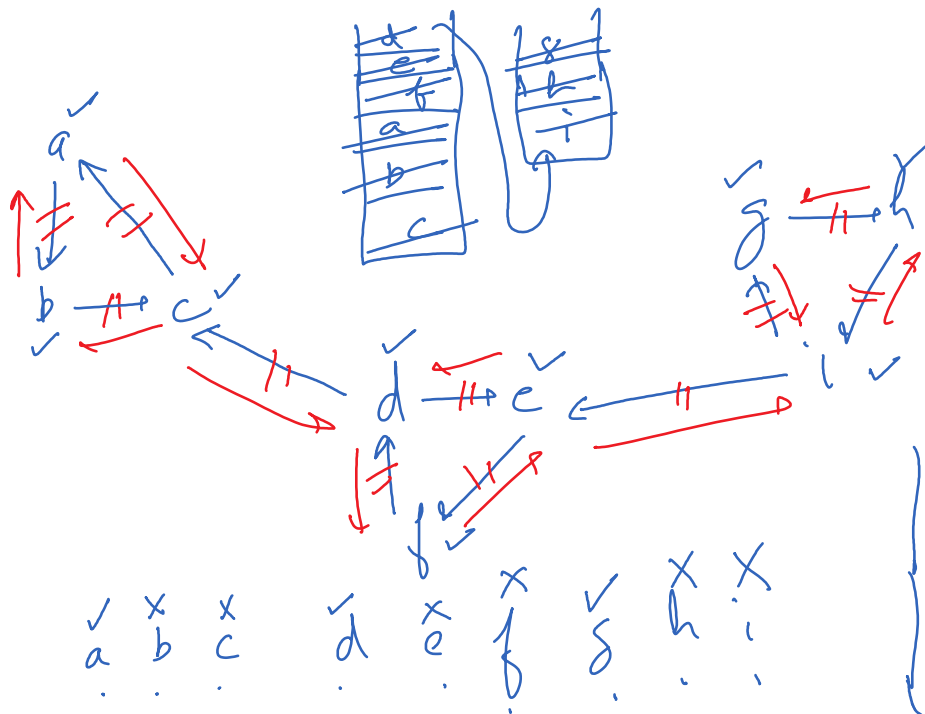
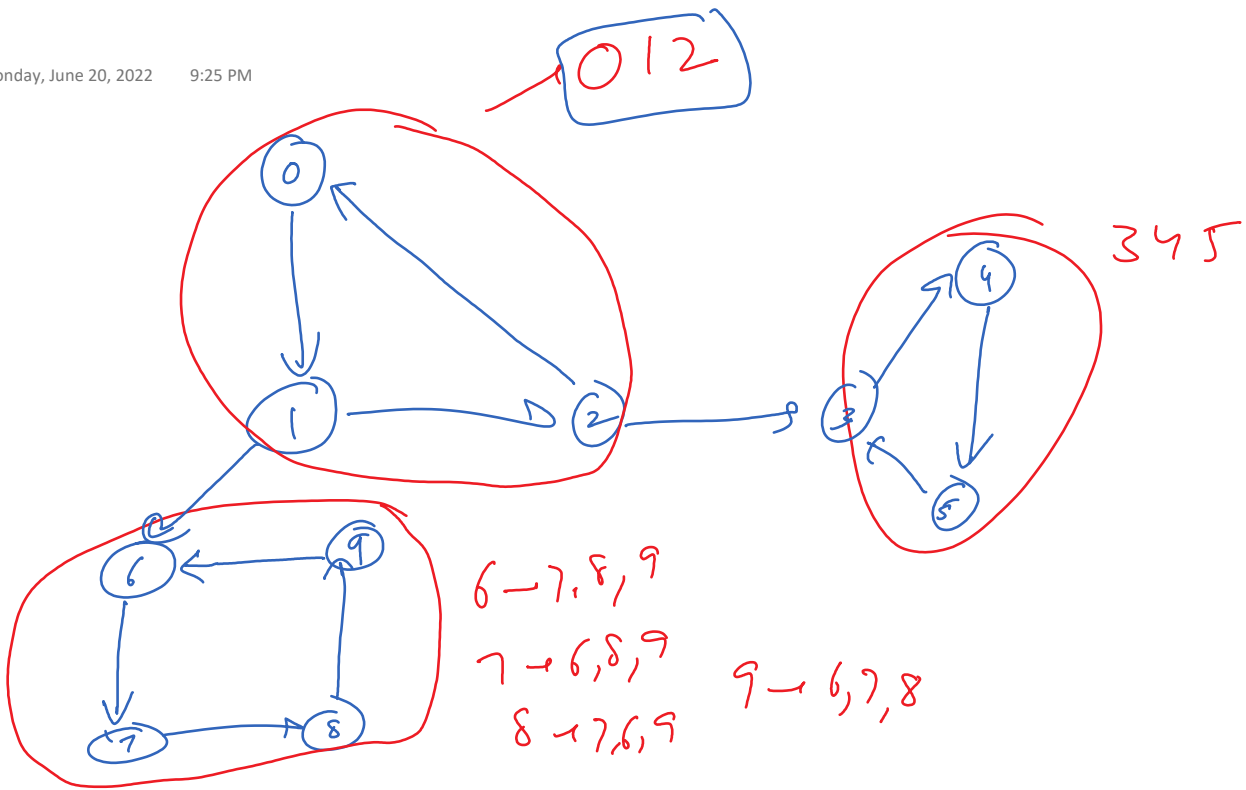




10:53 - 11:05

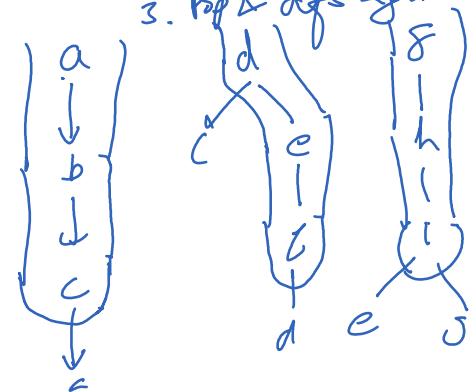






Kos = nej u

1. dfs from all
 ✓ & add to
 stack in post order
2. Transpose
3. Pop & dfs again



Kosaraju Algorithm.

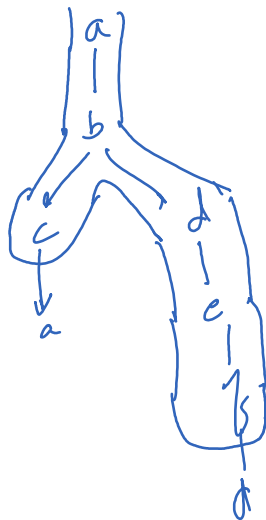
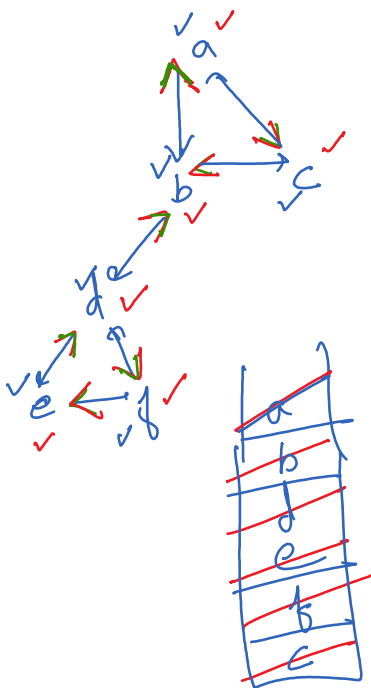
1. Dfs from all vces as root, add to st. in postorder.

2. Transpose

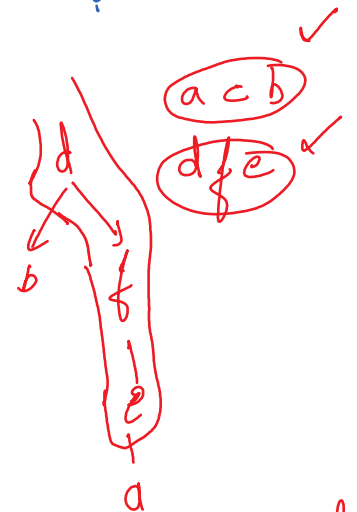
3. pop from st, do dfs with popped vertex as root

scc

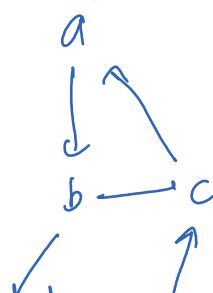
a b c d e f



Submit



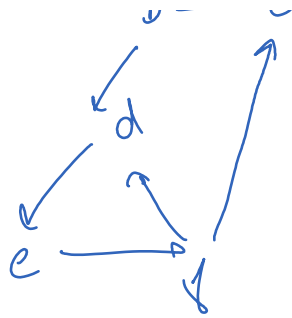
W1? Stack postorder.
W2? Transpose



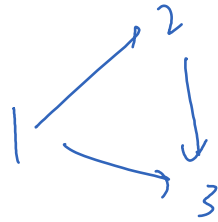
1. KR
2.
3. Submit

why?

9:56 - 10:10



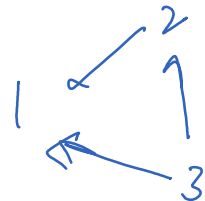
1:56 - 10:10



1 → 2, 3

2 → 3

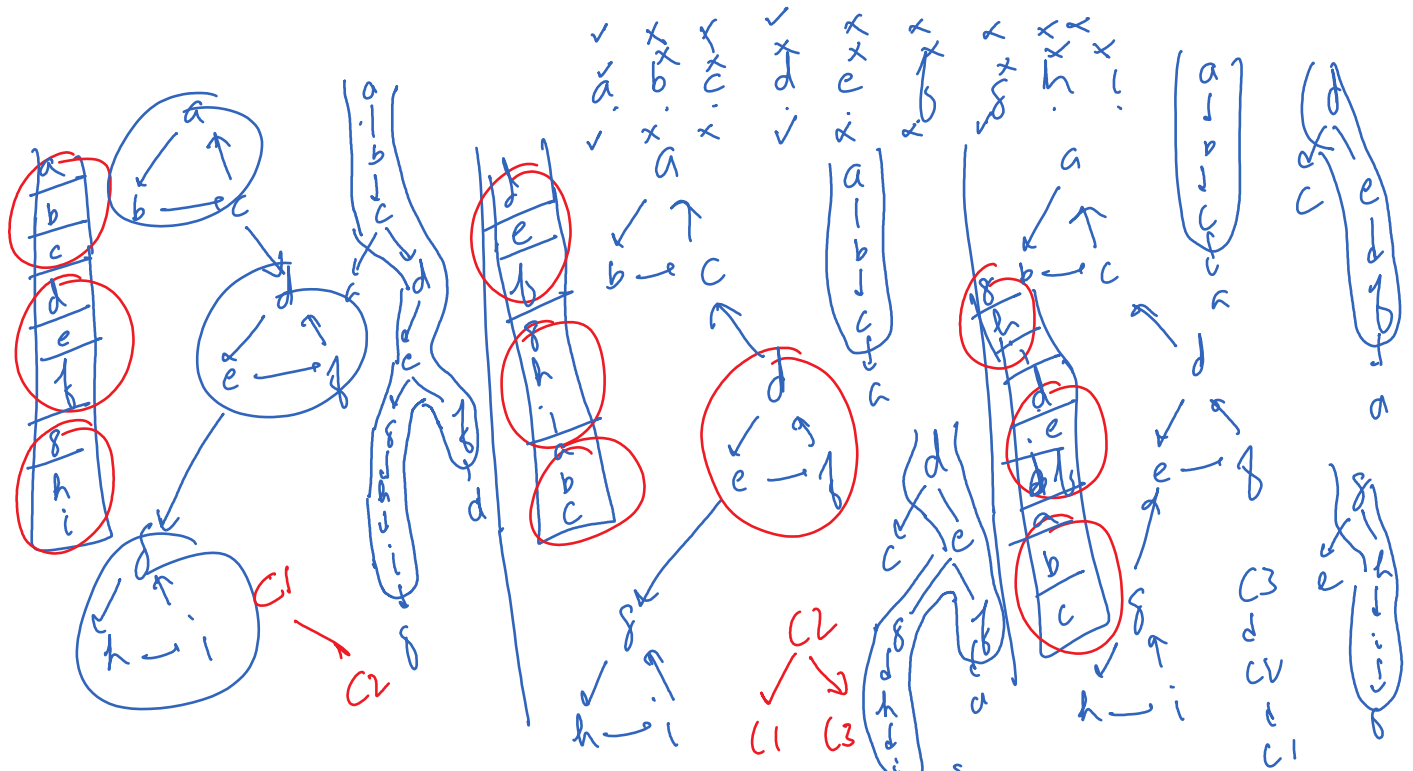
3 → []

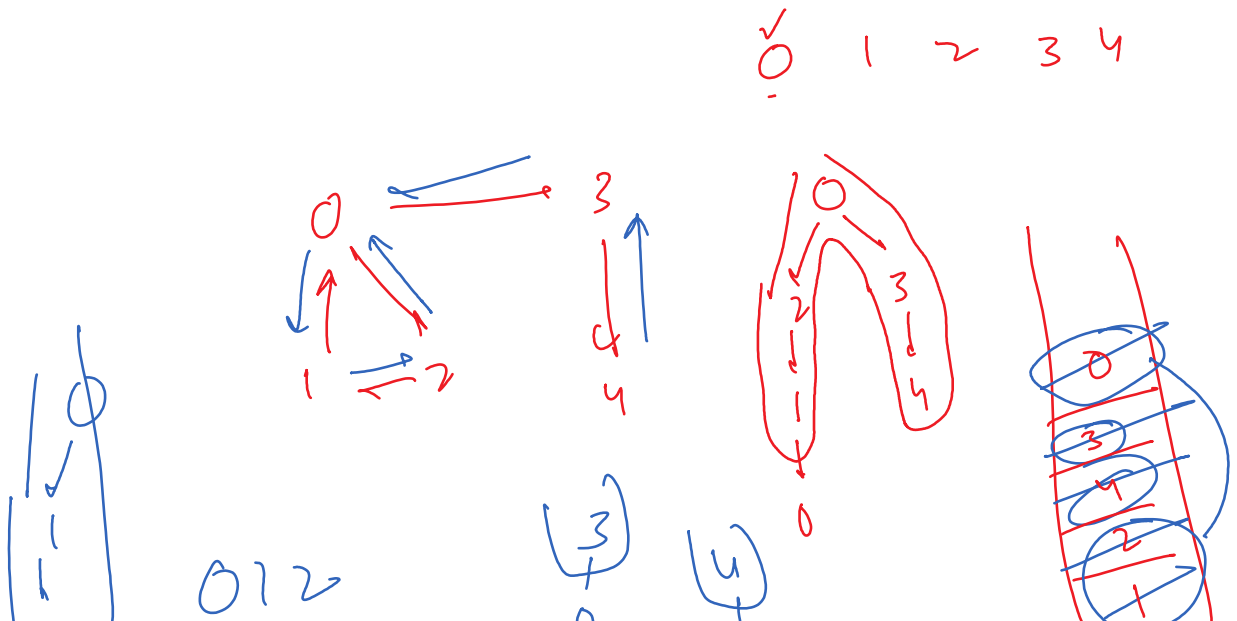
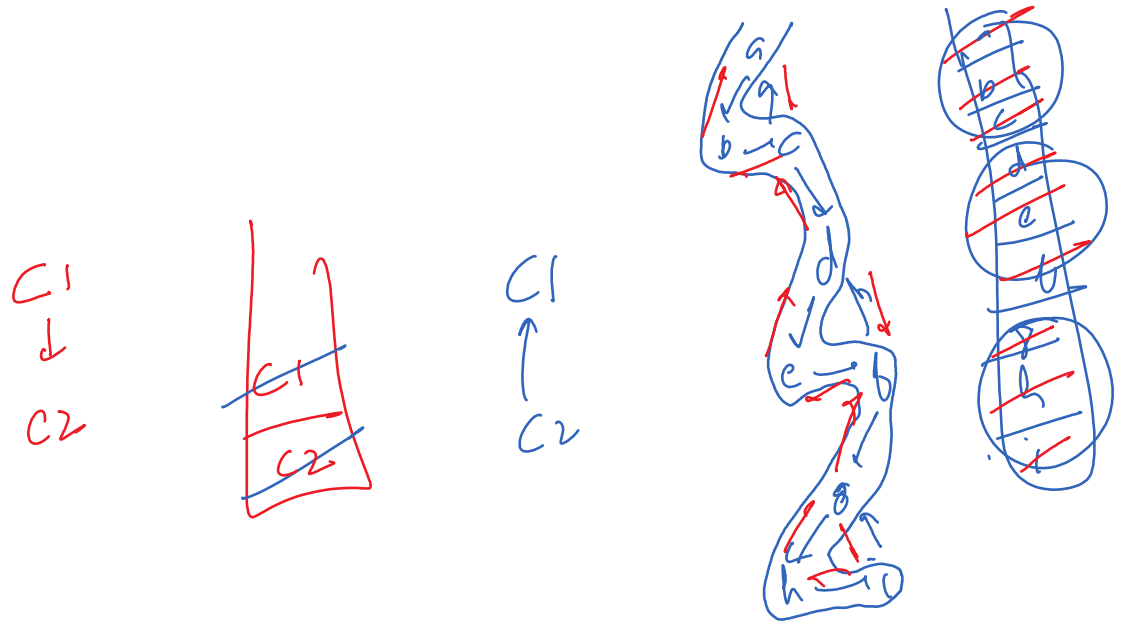
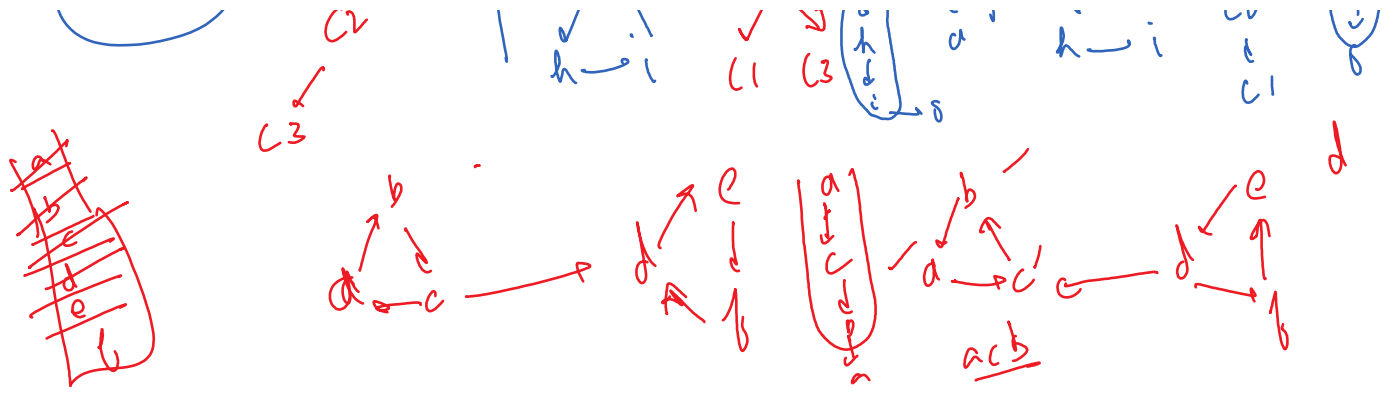


1 → []

2 → 1

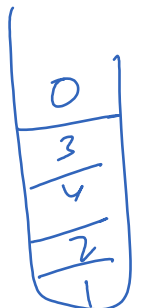
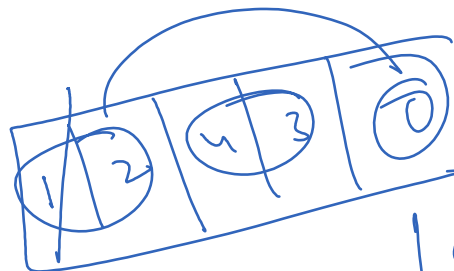
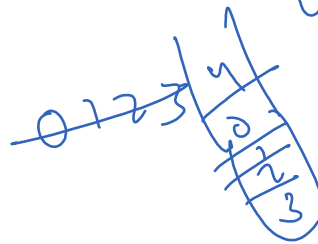
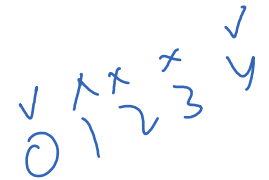
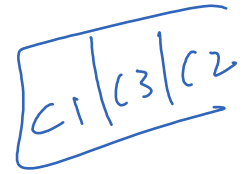
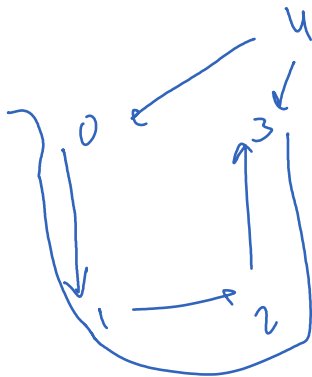
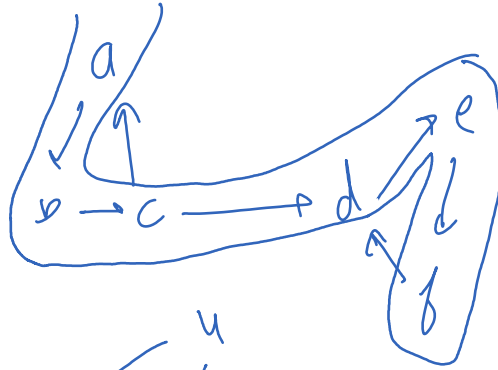
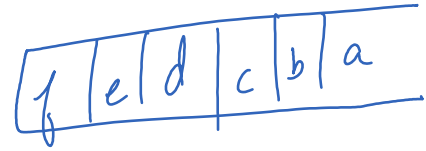
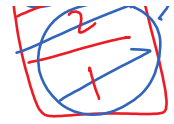
3 → 1, 2







0 1 2
3
4



10:56 - 11:06

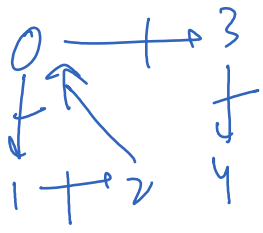
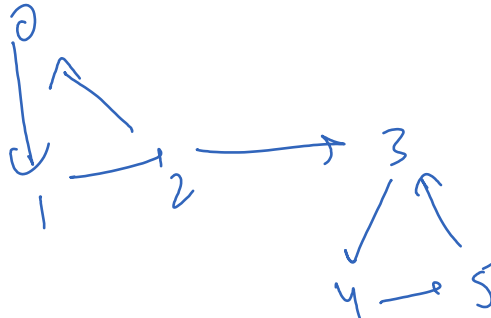
→ Momen Vertex

./ . v . ~ v

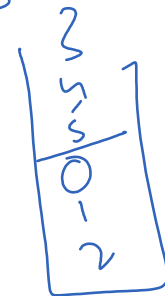
✓

1001

✓ 0 1 x 2 x 3 x 4 x 5

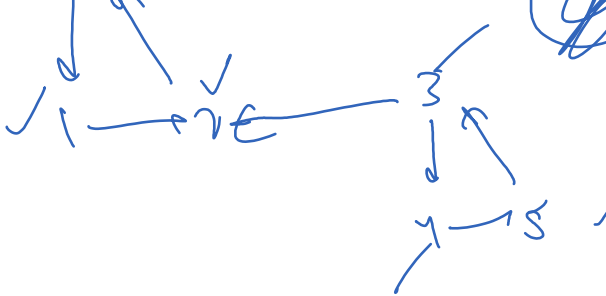


KR WS TS



KR = ○ → ○ 4 5

✓ 0 1 x 2 ✓ 3 4 5 ✓ 0



✓ 0 1 ✓ 2 ✓ 3 ✓ 4 ✓ 5



3



3

0' - [1' | 3']

1' - [0' | 2']

2' - [1' | 3']

3' - [0' | 5' | 4']

4' - [3' | 5' | 6']

5' - [4' | 6']

6' - [4' | 5']

1'

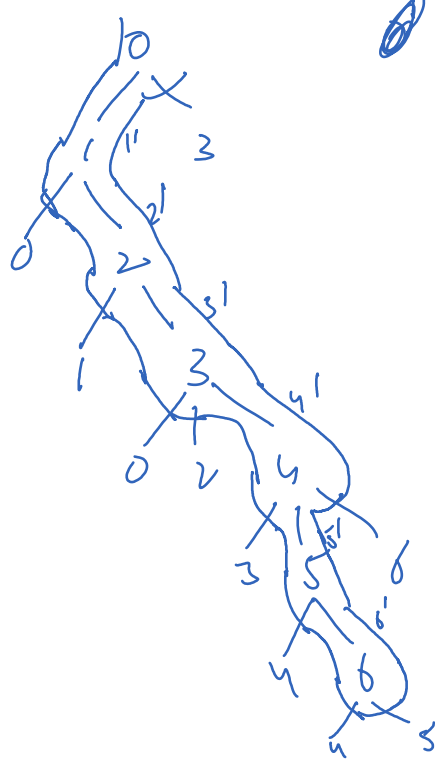
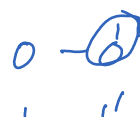
2'

5'

6'

```
public Node helper(Node node, Node[] visited){
    Node nodeClone = new Node(node.val);
    visited[node.val] = nodeClone;
    for(Node nbr: node.neighbors){
        if(visited[nbr.val] == null){
            Node nbrClone = helper(nbr, visited);
            nodeClone.neighbors.add(nbrClone);
        } else {
            Node nbrClone = visited[nbr.val];
            nodeClone.neighbors.add(nbrClone);
        }
    }
    return nodeClone;
}
```

0' - 1', 3'



0 - 0'
1 - 1'
2 - 2'
3 - 3'
4 - 4'
5 - 5'
6 - 6'

```

    }
    return nodeClone;
}

```

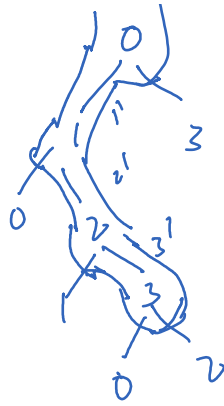
0' - 1', 3'

1' - 0', 2'

2' - 1', 3'

3' - 0', 2'

~~0' - 1', 3'~~



0 - 0'

1 - 1'

2 - 2'

3 - 3'

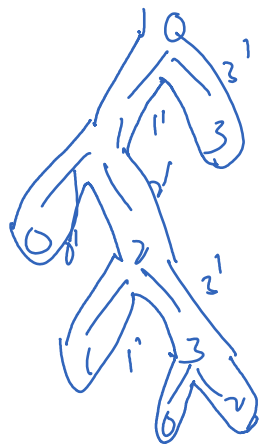
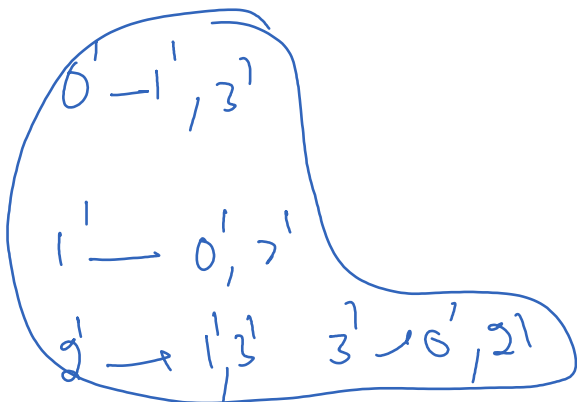
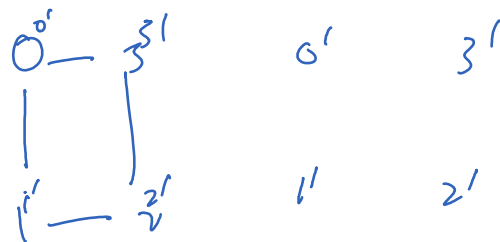
```

public Node helper(Node node, Node[] visited){
    if(visited[node.val] != null){
        return visited[node.val];
    } else {
        Node nodeClone = new Node(node.val);
        visited[node.val] = nodeClone;

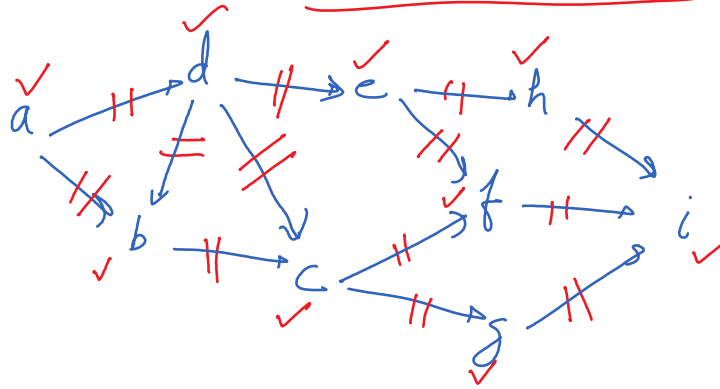
        for(Node nbr: node.neighbors){
            Node nbrClone = helper(nbr, visited);
            nodeClone.neighbors.add(nbrClone);
        }

        return nodeClone;
    }
}

```

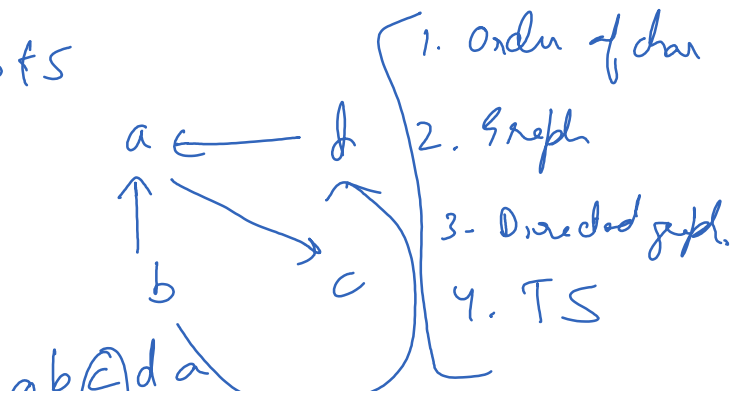
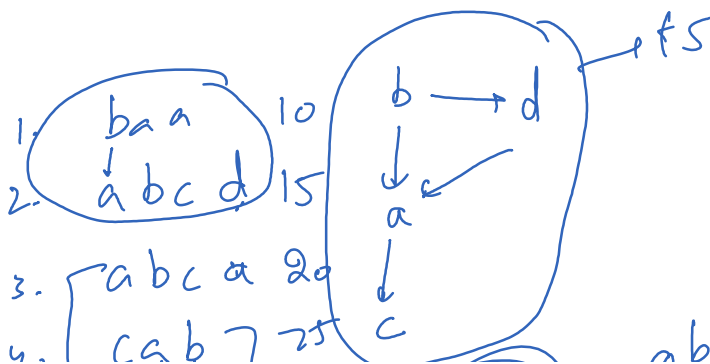
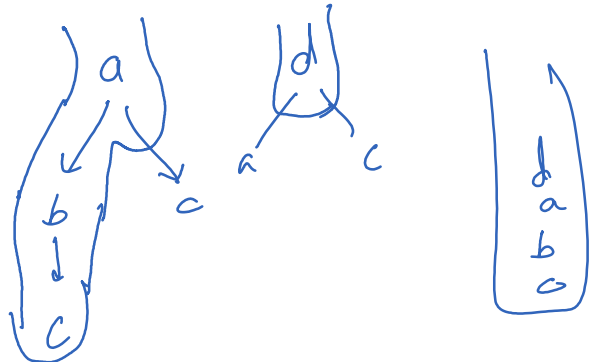
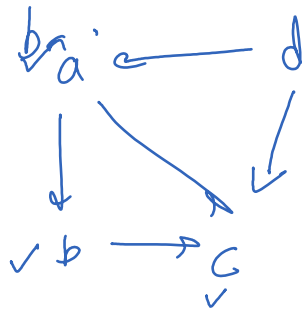


a d e h b c g f i



✓ a b c d e f g h i

✓ x x ✓
a b c d



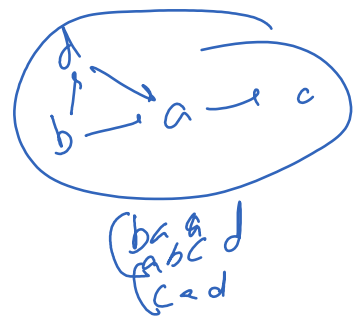
C

- 3. [a b c a] 29
- 4. [c a b] 25
- 5. [c a d] 30



7. 15

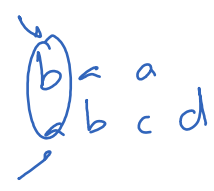
a: 54 - 10:10



a	0	→	
b	1	→	
c	2	→	
d	3	→	

d → a

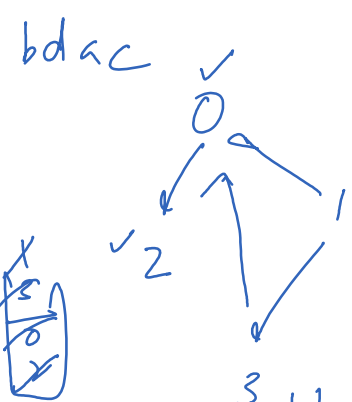
- ~~baa~~
- abcd
- abca
- cab -
- cad -



d → a

3 → 0

a	0	→	2
b	1	→	0 13
c	2	→	
d	3	→	0



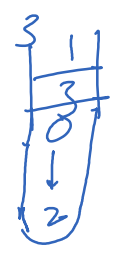
0 → a

1 → b

2 → c

3 → d

0 1 2 3



d → a

- baa -
- abcd
- abca
- cab -
- cad -

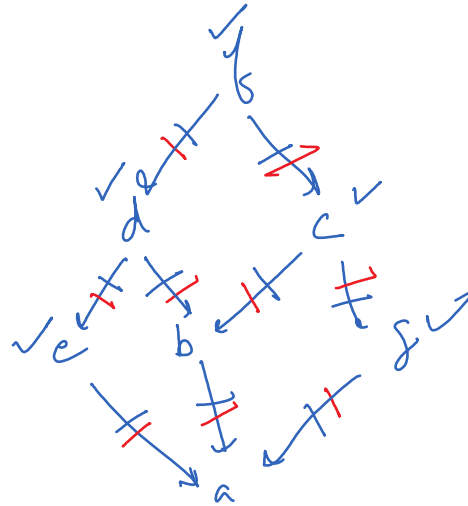
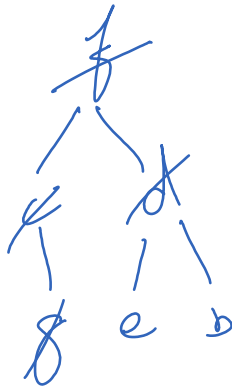
- abcd
- abca
- baa
- cab
- cad

HM < C, NS < C >

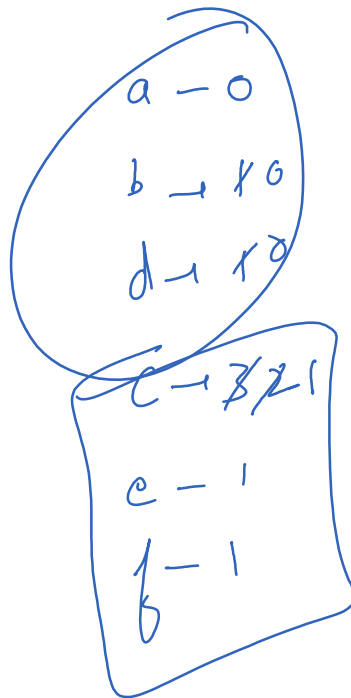
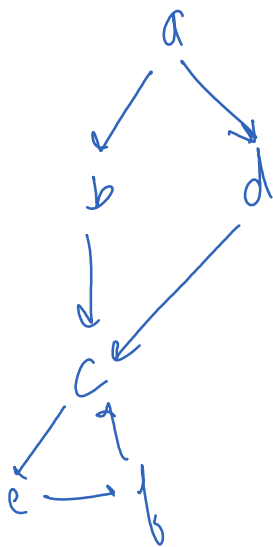
1111 /

f cd

a → 3
 b → 2 1 0
 c → 1 0
 d → 1 0
 e → 1 0
 ✓ f → 0
 g → 1 0

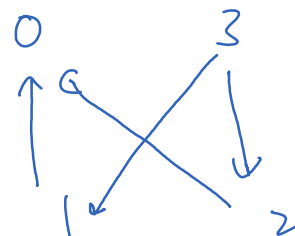


abd

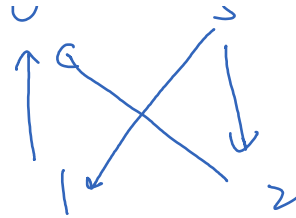


Input: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]

TS reverse



TS reverse

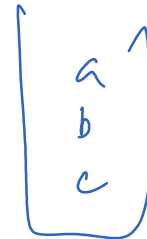
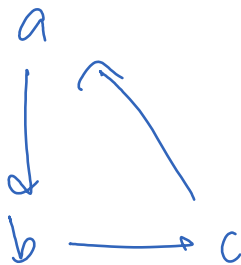


0 1 2 3

3 2 1 0

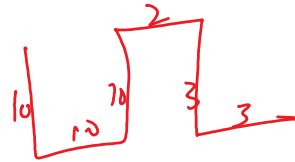
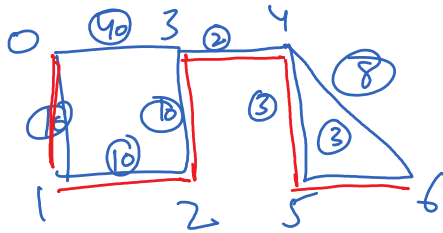
11:34 - 11:50

a b c



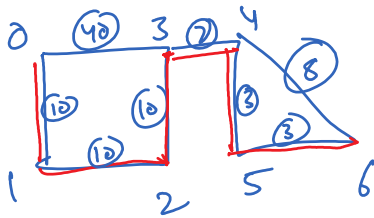
T - C > A C

kruskal [MST]
DSU



3-4	2	✓
4-5	3	✓
5-6	3	✓
4-6	8	X
0-1	10	✓
1-2	10	✓
2-3	3	✓
0-3	4	X

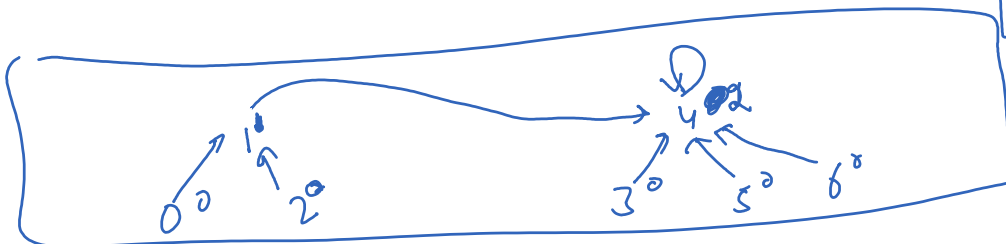
DS Union → [Find (PC) + Union (Rank)] application kruskal (MST)



3-4@2
6-5@3
4-5@3
0-1@10
3-2@10
1-2@10

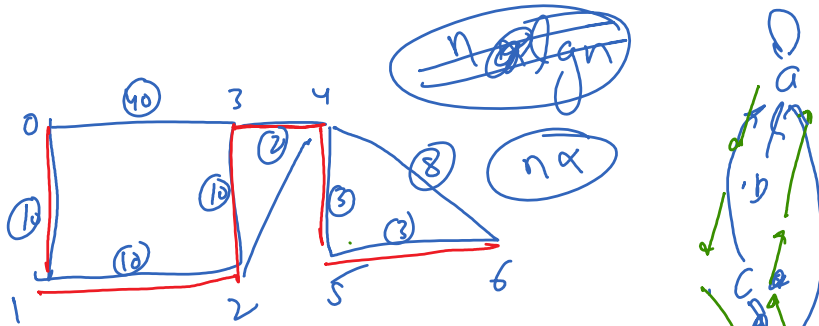
From <<https://pepcoding.com/resources/online-java-foundation/graphs/minimum-wire-to-connect-all-pcs-official/ojquestion>>

3-4	2	✓
4-5	3	✓
5-6	3	✓
4-6	8	X
0-1	10	✓
1-2	10	✓
2-3	3	✓
0-3	4	X



Prims → more edges
Kruskals → less edges

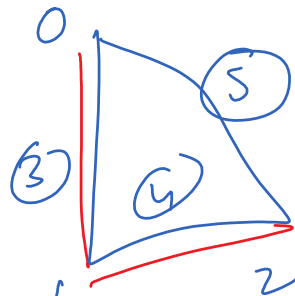
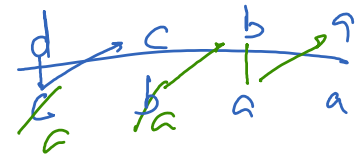
2nd iteration



3	4	2	2	✓
4	5	2	3	✓
5	6	2	3	✓
4	6	2	8	✗
6	7	2	10	✓
12	2	10		✓
23	2	10		✓
23	2	4	10	✗
24	2	5		✓



	0	1	2	3	4	5	6
p	3	3	1	3	3	3	3
n	0	1	0	2	0	0	0



0	1	2
0	1	2



10	23	✓
12	24	✓
22	25	✓

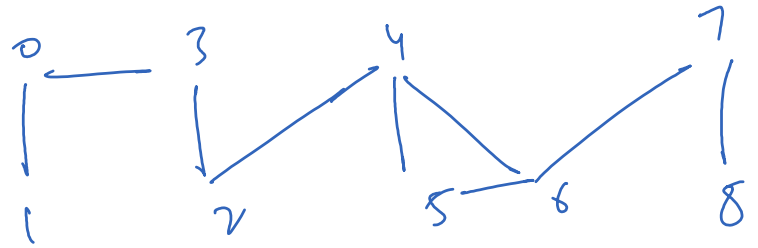
fructu

01 23
12 24

10:57 - 11:07

Graph Static \rightarrow BFS
DFS $(V+E)$

Graph Dynamic



BFS-DFS

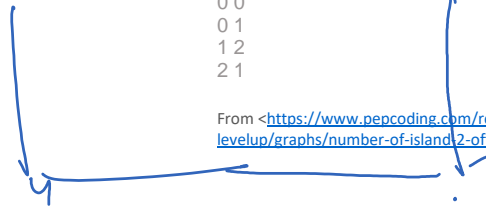
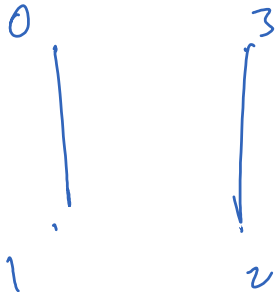
DSU

$\mathcal{O}(V+E)$

$\mathcal{O}(1)$

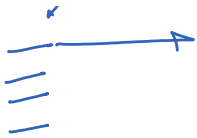
3 3 4
0 0
0 1
1 2
2 1

From <<https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/graphs/number-of-island2-official/ojquestion>>



9
8
7
6
5
4
3





11:27 to 11:37

	0	1	2
0	a 1	b 1	c 1
1	d 0	e 0	f 1
2	g 1	h 1	i 1

00
01



~~0~~
~~1~~



~~2~~ 1



2



3



~~4~~ 3



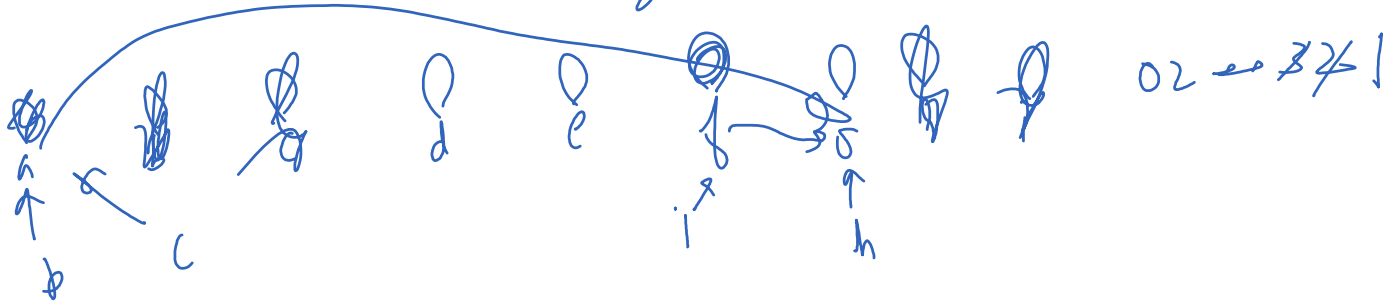
~~4~~ 2

12

20

21

22



	1		
j	1		1
o	1	1	1
1	1		

3 4

~~0~~
~~1~~
~~2~~
~~3~~
~~4~~
~~5~~ 3
~~4~~ 3
4
~~5~~ 2
~~3~~ 1
~~0~~

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19

$$4 \times 5$$

$$m \times n$$

$$bno = n \times n + c$$

$$2 \times 5 + 3$$